Restricted Boltzmann Machines

Mykola LIASHUHA, Vaibhav ARORA

1. Introduction

In this work we implement Restricted Boltzmann Machines (RBMs) with continuous observed variables and binary latent variables. The inference task is finding a parameters setting for the probabilistic model to capture the underlying data distribution for a custom toy dataset.

2. RBMs

Boltzmann Machines (BMs) are generative models and Probabilistic Graphical Models (PGMs) Boltzmann Machines. They model a family of distributions called Boltzmann distribution over a set of binary variables. The distribution itself is represented by an energy function. This energy function $E(y, x)$ returns a scalar energy value for each configuration of model parameters θ. BMs are a 2-layer network and consist of a visible (or observation) layer $\mathbf{x} = [x_1, x_2, ..., x_d] \in \mathbb{R}^d$ and a hidden layer $y = [y_1, y_2, ..., y_d] \in \mathbb{R}^p$. The visible layer is the layer that we can see; for example, it can be the layer of data. The hidden layer is the layer of latent variables which represent meaningful features or embeddings for the visible data [2]. Another interpretation is that in BMS, stochastic, binary data values are connected to stochastic, binary feature detectors using symmetrically weighted connections [1]. BMs have weighted links between two layers of neurons as well as links between neurons of every layer. RBMs restricts these links to not have links between neurons of a layer.

The following section is from [1] and finding it to be the most succint description, thus we we mention part of it here for sake of completion. Consider a joint configuration (y, x) of the visible and hidden units. Then a 2-layer network, a RBM can be trained to assign an approximate probability to every possible pair of a visible and hidden vector via the energy function,

$$E(y, x) = -\sum_{i \in visible} a_i x_i - \sum_{j \in hidden} b_j y_j - \sum_{i,j} x_i y_j w_{ij}$$

as:

$$p(y, x) = \frac{1}{Z} \exp^{-E(y,x)}$$

where the partition function Z is a normalizing term given by summing over all possible pairs of visible and hidden vectors,

$$Z = \sum_{y,x} \exp^{-E(y,x)}$$

Then the probability that the network assigns to a visible vector v is given marginalizing over all the hidden vectors:

$$p(v) = \frac{1}{Z} \sum_{h} \exp^{-E(y,x)}$$

What we want to do is adjust the weights of the network such that the probability it assigns to a data sample from the training distribution is raised (by lowering the energy of that image) and to lower the probability for other data (by increasing the energy).

The derivative of a training vector with respect to the weights can be written as [2]:

$$\frac{\delta \log p(x)}{\delta w_{i,j}} = \sum_{i=1}^{n} \mathbb{E}_{\sim p(y|x_i)}[\nabla_w(-E(y, x_i))] - n\mathbb{E}_{\sim p(y,x)}[\nabla_w(-E(y, x))]$$

The first term is referred to as the positive statistic and the second term as the negative statistic. We can simply write the equation as,

$$\frac{\delta \log p(x)}{\delta w_{i,j}} =< x_i y_j >_{data} - < x_i y_j >_{model}$$

where the angle brackets represent expectations under the distribution specified by the subscript. $< x_i y_j >_{data}$ is the expected value under the model of the product of hidden unit j and visible unit i when y is clamped to y_n and $< x_i y_j >_{model}$ is the expected number of times that x_i and y_j are both on if we sample from the model.

This leads to a simple learning rule for stochastic gradient ascent in the log probability of the training data for a learning rate ϵ:

$$\nabla w_{i,j} = \epsilon(< x_i y_j >_{data} - < x_i y_j >_{model})$$

3. RBMs interesting as Energy based models

As mentioned in the previous section, BMs have weighted links between two layers of neurons as well as links between neurons of every layer. RBMs restricts these links to not have links between neurons of a layer and form a bipartite graph. Therefore in RBMs, given the visible variables x, the hidden variables y are conditionally independent and vice-versa.

Thanks to this conditional independence, we can do the sampling for each unit in parallel in each layer. Moreover, Maximum likelihood learning is intractable in these models, but we show that learning can be performed efficiently by following an approximation to the gradient of a different objective function called "Contrastive Divergence". In fact, RBMs were one of the top entry in the Netflix Prize [3].

This can be put concretely as follows [1]: because there are no direct connections between visible units in an RBM, it is also very easy to get an unbiased sample of the state of a visible unit, given a hidden vector

For the positive statistic $< x_i y_j >_{data}$ we are conditioning on x_i, so we can take it out of the expected value. Because there are no direct connections between hidden units in an RBM, it is very easy to get an unbiased sample of $< x_i y_j >_{data}$. Given a randomly selected training sample,x, the binary state, y_j , of each hidden unit, j, is set to 1 with probability,

$$p(y_j = 1|x) = \sigma(b_j + \sum_i x_i w_{ij}$$

$x_i y_j$ is then an unbiased sample.

Similarly, because there are no direct connections between visible units in an RBM, it is also very easy to get an unbiased sample of the state of a visible unit, given a hidden vector,

$$p(x_i = 1|y) = \sigma(a_i + \sum_j y_j w_{ij}$$

Getting an unbiased sample of $< x_i y_j >_{model}$, however, is much more difficult. It can be done by starting at any random state of the visible units (usually setting the states of the visible units to a training vector) and performing alternating Gibbs sampling for a very long time. One iteration of alternating Gibbs sampling consists of updating all of the hidden units in parallel using the equation of $p(y_j = 1|x)$ followed by updating all of the visible units in parallel using the equation $p(x_i = 1|y)$. Therefore, With M true samples (x_m, y_m) from the distribution defined by the RBM, we could approximate $< x_i y_j >_{model}= \frac{1}{M} \sum_{m=1}^{M} x_{mi} y_{mj}$ We can get these samples by initializing N independent Markov chain at each data point x_n and running until convergence. Again, the type of Markov transition operator used most often is alternating Gibbs.

RBMs typically learn better models if more steps of alternating Gibbs sampling are used before collecting the statistics for the second term in the learning rule, however a single step is used for practical reasons with good enough approximation (Contrastive Divergence algorithm). The idea behind contrastive divergence is to run the Markov chain for only one step, get samples $(x_n^1; y_n^1)$, and hope that the model converges.

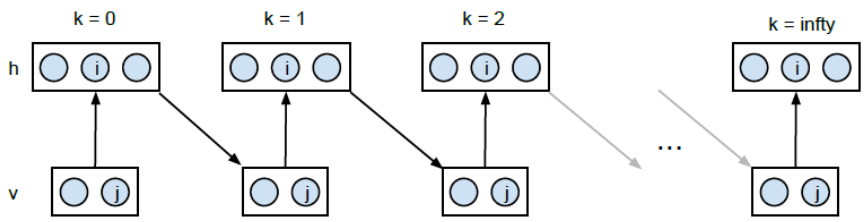
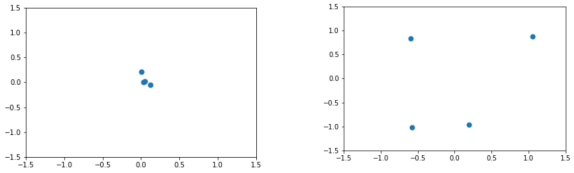


Figure 1: Gibbs sampling procedure in RBMs

4. Model space exploration when using a single Markov Chain to sample from the model

Common algorithms for training a RBM model approximates the second expectation term with respect to the model to do the gradient ascent step. Typical approach is using the MCMC methods to approximate this gradient, but it requires many steps until reaching the equilibrium state. As mentioned earlier, Contrastive Divergence (CD) is a standard way to train a RBM model. The idea is running k steps Gibbs sampling until convergence but $k = 1$ for CD.



(a) Single step (b) k steps

Figure 2: Gibbs samples

As we can see from Figure 2, with a single step, the model space is explored close to the previous samples compared to the space exploration with k-steps. The trade-off lies in computation time and convergence time.

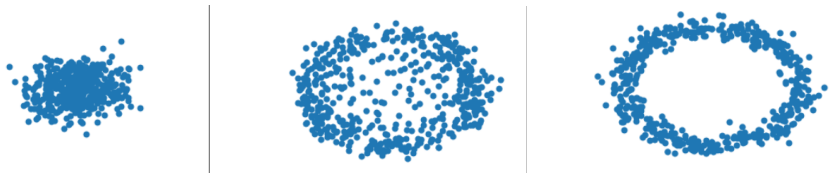
5. Practical advice for someone who needs many samples from the model

During sampling process we apply a method to draw samples from pretty unusual joint probability distribution such as Gibbs distribution. Since in RBM we have such an usual distirbution we have to apply Mrkov chain in order to approximate values. The pipeline implies drawing samples from $p(x|y)$ (visible value sample) and $p(y|x)$ (hidden value sample) sequentially for k-times. During sampling process the new sampled value is drawn taking into account the value from previous iteration. Though in the beginning of sampling process it is common to sample the hidden value first with pre-initialized parameters of hidden state distribution. For example, in case hidden state distribution is Bernoulli one, we intend to sample first hidden value from Bernouli distribution with μ parameter of 0.5.

$$z^1 \sim Bernoulli(0.5)$$
$$x^1 \sim p(x|z^1)$$
$$z^2 \sim p(z|x^1)$$
$$x^2 \sim p(x|z^2)$$
$$.$$
$$.$$
$$x^k \sim p(x|z^k)$$

As it is shown in the formulas, every next sample is dependent on the previous one. That is why it is important to take into consideration number of k iteration steps that would completed in order to achieve a well-generated sample. As an example,

below there are images provided to display generated points (from multiple chains) at different interaction step. At final step, all points should form a particular circle.



(a) 1 iteration (b) 2 iterations (c) 10 iterations

Figure 3: Generated data

Besides, it is important to note that there is a particular effect called "burn-out". During sampling process only a few first iterations the generated samples improve a lot. However, after the burn-out, the samples are approximately samples from the joint distribution P(x, z).

6. Results

In this lab exercises the main task was to implement from scratch Restricted Boltzmann Machine using PyTorch. The RBM model should learn the generative process according to provided dataset. The dataset is provided as circle-shaped areas of points.

In the model the hidden state was represented with Bernoulli distribution and visible one with Gaussian. For visible we have chosen Gaussian since our data is continious). Below there are our formulas that we used for sampling the hidden and visible representation:

$$z \sim Bernoulli(\mu),\ where\ \mu = sigmoid(\frac{x}{\sigma^2}W + d)\ and\ \sigma\ is\ const.$$

$$x \sim Gaussian(\mu, \sigma),\ where\ \mu = zW^T + b\ and\ \sigma\ is\ const.$$

Also, we used log-partition function $c(x;\omega)$ in order to calculate the derivative. The final loss function is calclated as difference between log-partition of target value and log-partition of generated one.

$$c(x;\omega) = \sum_{2}^{i=1} \frac{(x_i - b_i)^2}{2\sigma^2} - \sum_{n}^{j=1}[1 + exp(d_j + \sum_{2}^{i=1} \frac{x_i}{\sigma^2} W_{ij})]$$

Below are the results of training.

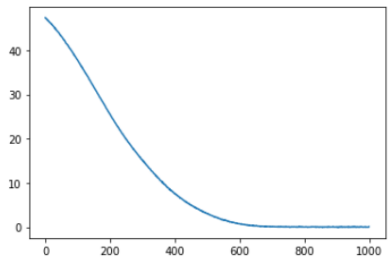


Figure 4: Training loss

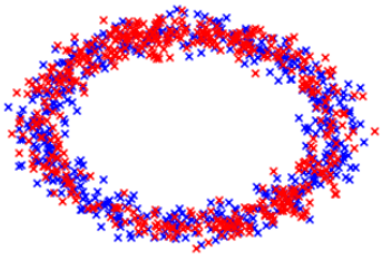


Figure 5: Target and generated values: blue dots are target ones, and red ones are generated

Overall, our model is performing well, and the generated samples are copying size/shape of target ones.

References

[1] Hinton G.E. A practical guide to training restricted boltzmann machines, 2012. 1, 2

[2] Benyamin Ghojogh, Ali Ghodsi, Fakhri Karray, and Mark Crowley. Restricted boltzmann machine and deep belief network: Tutorial and survey, 2021. 1

[3] Geoffrey Hinton Ruslan Salakhutdinov, Andriy Mnih. Restricted boltzmann machines for collaborative filtering, 2007. 2